ANALISIS ALGORITMA

- Algoritma adalah urutan langkah yang tepat dan pasti dalam memecahkan suatu masalah secara logis.
- Beberapa masalah dapat diselesaikan dengan algoritma yang bermacam macam asal hasilnya sama.
- Setiap bahasa pemrograman memiliki kelebihan dan kekurangan dalam mengimplementasikan algoritma dan setiap pemrogram dapat mengimplementasikan suatu algoritma dengan cara yang berbeda-beda pula.
- Namun algoritma dapat dianalisis efisiensi dan kompleksitasnya.
- Penilaian algoritma didasarkan pada:
  - Waktu eksekusi (paling utama)
  - Penggunaan memori/sumber daya
  - Kesederhanaan dan kejelasan algoritma
- Analisis algoritma tidak mudah dilakukan secara pasti, maka hanya diambil:
  - Kondisi rata-rata (*average case*)
  - Kondisi terburuk (*worst case*)
- Waktu eksekusi dipengaruhi oleh:
  - Jenis data input
  - Jumlah data input
  - Pemilihan instruksi bahasa pemrograman
- Faktor-faktor yang menyulitkan analisis disebabkan oleh:
  - Implementasi instruksi oleh bahasa pemrograman yang berbeda
  - Ketergantungan algoritma terhadap jenis data
  - Ketidakjelasan algoritma yang diimplementasikan
- Langkah-langkah analisis algoritma
  - Menentukan jenis/sifat data input.
  - Mengidentifikasi *abstract operation* dari data input.
  - Menganalisis secara matematis untuk menentukan *average case* atau *worst case* nya.

NOTASI Big-Oh

adalah ***fungsi*** yang lebih berkaitan dengan :

- ✓ kelajuan proses dari pada
- ✓ kelajuan pertambahan data.

$T(n) = O(f(n))$

jika ada konstan c dan no sedemikian rupa sehingga

$T(n) \leq c.f(n)$ untuk $n \geq no$

Secara sederhana dikatakan bahwa $O(f(n))$ dpt dianggap sebagai nilai maksimum dari $c.f(n)$.

Contoh:

Jika $f(n) = 2n^2$ maka $T(n) = O(n^2)$

Jika $f(n) = 4n^2 + 7n + 3$ maka $T(n) = O(n^2)$

Contoh program:

```
sum = 0;
for(i=0; i<n; i++)
     sum = sum + a[i];
```

Penghitungan eksekusi:

° sum = 0	dieksekusi 1 kali
° i = 0	dieksekusi 1 kali
° i < n	diekseklusi n kali
° i++	dieksekusi n kali
° sum = sum + a[i]	dieksekusi n kali
	----------
	2 + 3n kali

Jadi T(n) = O(n)

Contoh program:

```
sum = 0;
for( i=0; i<n; i++)
     for(j=0; j<n; j++)
          sum = sum + a[j];
```

Penghitungan eksekusi:

- sum = 0 dieksekusi 1 kali
- i = 0 dieksekusi 1 kali
- i < n dieksekusi 1 kali
- i++ dieksekusi n kali
- j = o dieksekusi n kali
- j < n dieksekusi n × n kali
- j++ dieksekusi n × n kali
- sum = sum + a[j] dieksekusi n × n kali

$3 + 2n + 3n^2$ kali

➔ T(n) = **$O(n^2)$**

KLASIFIKASI ALGORITMA

T(n) = c

- Jika sebagian besar instruksi pada program hanya dieksekusi *constant* kali.
- Bersifat konstan, tidak bergantung pada parameter atau banyak data yang diolah
- Merupakan algoritma paling ideal

Contoh:

```
void hitung()
{
     a = 6;
     b = 4;
     c = a + b;
}
```

T(n) = O(n)

- Waktu eksekusinya sebanding/sama dengan jumlah data.
- Setiap data input hanya diolah secara sederhana. Misal hanya di tambah,
- dikurangi, dikalikan, dan dibagi.
- Inilah kondisi optimal yang dicari dalam membuat algoritma.

Contoh :

```
for(int t=0;  t < n ;  t++)
  a[t] = 0;
```

T(n) = $O(n^2)$

- Waktu eksekusi berbading dengan kwadrat jumlah data. Misal:
  - Jika n=10 waktu eksekusinya 100
  - Jika n=20 waktu eksekusinya 400
- Biasanya timbul karena setiap data dieksekusi 2 kali, misal dlm *nested loop*.
- Contoh : Algoritma Bublesort

Contoh:

```
for(int i=0 ; i<n ; i++)
     for(int j=0 ; j<m ; j++)
          a[i,j] = 0;
```

Loop luar dieksekusi n kali
Loop dalam dieksekusi m kali
Total eksekusi = n × m kali

T(n) = O(log n)

- Waktu eksekusi sebanding dengan logaritma dari jumlah data.

- Misal jumlah data menjadi 2 kali semula berarti waktu proses bertama 1 unit, jika 1000 kali bertambah 10 unit, jika sejuta berarti 20 satuan.
- Biasanya untuk algoritma yang memecah masalah besar kedalam sub masalah yang lebih kecil.
- Contoh : Algoritma *binary search* dan algoritma fungsi rekursif

$T(n) = O(n \log n)$

- Bersifat linieritmis (linier dan logaritmis)
- Untuk algoritma yang memecah masalah besar menjadi sub-masalah yang lebih kecil dan diselesaikan secara terpisah, kemudian hasilnya digabungkan.
- Contoh : Quicksort

$T(n) = O(n^3)$

- Untuk algoritma yang mengolah setiap data input 3 kali. Biasanya berupa 3 buah *nested loop*.
- Algoritma semacam ini sebisa mungkin dihindari.

Contoh:

```
for(i=0 ; i<n ; i++)
     for( j=0 ; j<n ; j++)
          for(k=0; k<n; k++)
               a[i,j,k] = 0 ;
```

n	log n	n log n	n2	n3
10	3	30	100	1000
100	6	600	10000	100000
1000	9	9000	1000000	1000000
10000	13	130000	100 juta	1 trilyun
100000	16	1600000	10 milyard	100 trilyun

- Pada implementasinya, algoritma kadang-kadang memiliki Big-Oh yang merupakan kombinasi dari klasifikasi dasar tersebut.
- Perancangan algoritma yang memiliki Big-Oh baik tetapi rumit tidak selalu menguntungkan apabila jumlah data inputnya hanya sedikit.
- Nilai Big-Oh adalah nilai *worst case*, sehingga dalam implementasinya biasanya tidak mencapai nilai setinggi itu.

CONTOH DAN ATURAN

1. For loop (perulangan)
   Waktu eksekusi pada *for loop*, maksimum sebanyak waktu eksekusi *statement-statement* yang ada di dalam *loop* dikalikan banyaknya iterasi.
   Contoh:
   ```
   for(a=0; a<n; a++)
   {
    m = p + q;
    t = y * z;
   }
   ```
   Waktu eksekusi = 2 x n kali
   Jadi T(n) = O(n)
2. Nested for loop (perulangan bersarang)
   - Dianalisis dari loop terdalam kemudian keluar.
   - Waktu eksekusi total sebuah statement adalah waktu eksekusi statement tsb dikalikan hasil kali dari banyaknya iterasi semua *loop* yang ada diluarnya.

   Contoh:
   ```
   for(i=0 ; i<n ; i++)
    for( j=0 ; j<m ; j++)
       a[i,j] = 0 ;
   ```
   a[i,j] akan dieksekusi sebanyak (m x n) kali.

Jadi T(n) = O(n2)

3. Consecutive Statement (statement yang berurutan)
   - Untuk statement yang berurutan, waktu eksekusinya adalah jml dari masing-masing statement.
   - Berdasarkan pengertian Big-OH, hal tsb akan diambil nilai yang terbesar.

   Contoh:

```
for(k=0; k<10; k++)                          { T1(n) = O(n2) }
     x[k] = 0;
     for(i=0 ; i<n ; i++)                    { T2(n) = O(n2) }
          for( j=0 ; j<m ; j++)
               a[i,j] = 0 ;
```

   Jadi T(n) = T1(n) + T2(n) = O(n2)

4. if else

   Total waktu eksekusi adalah waktu *test* ditambah waktu yang terbesar ~~dari~~ eksekusi Statemen1 atau Statemen2

   Contoh:

```
if(a < 10)                    Waktu Tes = 1
{
 m = q + r;                   Waktu Tes = 2
 k = y * z;
}
else
{
 for(t=0; t<10; t++)          Waktu Tes = 10
      x = x + t;
}
```

   Jadi total waktu eksekusi adalah 1 + 10 = 11

   Jadi T(n) = O(n)

LATIHAN

```
int select(int a[], int k, int n)
{
     int i, j, mini, tmp;
     for(i=0; i<k; i++)
     {
          mini = i;
          for(j=i+1; j<n; j++)
               if (a[j] < a[mini])
                    mini = j;
               tmp = a[i];
               a[i] = a[mini];
               a[mini] = tmp;
          }
          return a[k-1];
     }
}
```